# EECS 214 Worksheet: Hash Tables

For this worksheet, you will be hashing values consisting of strings of characters `a` , `b` , and `c` . You will use a hash function *h(x)* defined as follows:

h(x) = *integer representation of string % number of buckets in table*

where integer representations of strings are computed by mapping each character to the corresponding value and taking the sum of all characters in the string:

```
“a" -> 11
"b" -> 22
"c" -> 33
```

Note that we are using the **division method** to convert the resulting integer into an index for the hash table, by taking the integer modulo (%) the number of buckets in the table. **For this worksheet, assume your hash table has 10 buckets.** Remember that modular arithmetic is essentially taking only the remainder, f.e. (27 mod 10 = 7).

> ***Example.*** *The input* `"aca"` *would map to the hash value 5.*
> *1. Convert to integer:* `"aca" -> 11 + 33 + 11 = 55`
> *2. Division method: mod the integer value by the number of buckets, in this case 10:* `55 % 10 = 5`

**Question 1.** For each of the following inputs *x*, compute the resulting hash value *h(x)*.

`"aaa"` →
`"abc"` →
`"bbc"` →
`"bcb"` →
`"acb"` →
`"cac"` →

**Question 2.** Assume you’re using an array to store the hash table (each cell of the array corresponds to one “bucket”). Now assume we insert the 6 inputs from Question 1 **in that order**.

If we were using **chaining**, what would the hash table look like after we inserted all 6 strings? (Draw out your linked lists if necessary.)

- **0:**
- **1:**
- **2:**

- **3:**
- **4:**
- **5:**
- **6:**
- **7:**
- **8:**
- **9:**

What would the hash table look like if we were using **open addressing with linear probing**?

- **0:**
- **1:**
- **2:**
- **3:**
- **4:**
- **5:**
- **6:**
- **7:**
- **8:**
- **9:**

**Question 3.**

1. What is the load factor of the chained hash table?

2. What is the load factor of the open addressed hash table?

3. Suppose after entering the above values into our open addressed hash table, we wanted to *look up* the string `"cac"`.
   a. How many iterations would it take to find the index of this string?
   b. What is the average case time complexity for this operation?
   c. Suppose we had 21 entries in our hash table. How might the number of iterations required to lookup a value change?

**Question 4.**

1. What are some **disadvantages** of linear probing?

   a. How might we mitigate these disadvantages?

2. What are some **disadvantages** of chaining?

**Sample Interview Questions:**

> *These are all questions that we have personally encountered on interviews. While there are other ways to solve these questions besides using hash tables, solutions using hash tables will have strong time complexities for these questions. Knowing how to use hash tables effectively will give you great returns on the computer science job hunt! (Some of us have used hash tables on interviews more times than we'd like to count)*
> *Note that we are not necessarily distinguishing between a chained hash table and an open addressed hash table for these problems.*

**Question 5.**

Using a hash table, find whether two inputted strings, *s1*, and *s2*, are valid anagrams of each other. Return true if true, and false otherwise. What would the time complexity be of this algorithm? If we didn't use a hash table, and instead simply iterated through each character of a string with all the characters of the other string, what would the time complexity be?

> *Note that an anagram occurs when the two strings have the same exact number of ocurrences for the same characters. I.e. falafel and llaaeff are anagrams of each other, but falafel and laef are not.*

**Question 6.**

Using a hash table, find whether an array contains at least duplicate element. What would the time complexity be of this algorithm? If we didn't use a hash table, and instead simply iterated through each character of a string with all the characters of the other string, what would the time complexity be?

> *Hint: try using a hash table which has a value of boolean type.*
> *An example input → output would be*
> *{1, 2, 3, 4, 5, 3} → true,*
> *{1, 2, 3} → false*
> *{1, 1, 2, 2 } → true*

**Question 7.**

Using a hash table, find whether an array contains a majority element, where a majority element is defined as an element which occurs more than half of the time.

> *Sample input → output:*
> *{1, 1, 1, 1, 2} → true*
> *{1, 2} → false*
> *{1, 1, 1, 2,3,4} → false (note it has to be* ***more*** *than half of the time)*
> *{2, 1, 2, 3, 2, 2,} → true*